AN4367
Application note

SPC56EL60xx/SPC56xL70xx device exception handling

Introduction

This document provides overview of SPC56EL60xx/SPC56xL70xx exception handling with main focus on different kind of exception that the application code can face during the runtime like Flash 2b ECC error, RAM 2b ECC error, MPU protection violation, AIPS access protection violation and others.

It starts simple overview of Machine check interrupt highlighting important things from application perspective. To get detailed view and to implement low level machine check interrupt handler it is necessary to use Z4 Core User Manual which describes all the details about the Core exception and interrupts.

The following part lists the SPC56EL60xx/SPC56xL70xx exceptions, describes the reason of the exception, how to find it and what possibilities exists to remove the fault.

Contents

1 Z4 Core exception overview 5
1.1 Machine check interrupt (IVOR1) 5
1.1.1 Machine check registers 5

2 Machine check handler 7
2.1 Low level handler 7
2.1.1 Start phase 7
2.1.2 Final phase 8
2.1.3 Modification of the MCSRR0 register 8
2.2 User handler 9

3 SPC56EL60xx/SPC56xL70xx exception cases 10
3.1 Flash 2b ECC error 10
3.1.1 Cause of the exception 10
3.1.2 Machine check exception status 11
3.1.3 User exception handler 11
3.1.4 Error solving 11

Revision history 12

List of tables

Table 1. Machine check interrupt causes 5
Table 2. Machine check register 5
Table 3. Machine check causes 6
Table 4. SPC56EL60xx/SPC56xL70xx exception causes 10
Table 5. Flash 2b ECC - machine check exception status 11
Table 6. Document revision history 12

List of figures

Figure 1. Machine check exception flow 7
Figure 2. Modification of MCSRR0 register content. 8
Figure 3. Machine check exception user handler flow 9
Figure 4. Flash 2b ECC error 10

1 Z4 Core exception overview

Z4 Core used on SPC56EL60xx/SPC56xL70xx devices contains many exception sources and sixteen interrupts to service them. Multiple exception sources can be mapped to one interrupt handler where few supportive status registers provide flags to find the cause of the exception in the handler.

Detail list of exception causes and their mapping to interrupt handlers can be found in the Z4 Core Reference Manual. This reference manual is available online at http://www.st.com.

This chapter gives overview simple overview of machine check interrupt that is utilized for several important fault states of SPC56EL60xx/SPC56xL70xx device.

1.1 Machine check interrupt (IVOR1)

Machine check interrupt is a handler that services multiple fault events that can occur during runtime code execution.

Table 1. Machine check interrupt causes

Interrupt type	Exception conditions
Machine check	– NMI – ISI, ITLB, Error on first instruction fetch for an exception handler – Parity Error signaled on cache access – External bus error

This interrupt is used to handle various faults generated by peripherals in the SPC56EL60xx/SPC56xL70xx device, like MPU protection fault, 2bECC error in the Flash or RAM memory etc. The reason is that most of the faults are signaled back as external bus error situation during the CPU-Submodule bus transaction.

1.1.1 Machine check registers

Z4 core implements few machine check status registers that are updated upon the exception event with some constraints stated in the Z4 Core Reference Manual. These registers are used to find the source of the exception and based on it to decide how to solve it.

Table 2. Machine check register

Register	Content
MCSR (syndrome register)	Register indicates the source of machine check condition that gives possibility to differentiate between them
MCAR (address capture)	Register contains for some sort of machine check conditions the address for which the asynchronous type of the machine check exception was raised. Address valid only when MCSR.MAV bit was ‘0’ before the exception, otherwise MCAR register is not updated.

Table 2. Machine check register (continued)

Register	Content
MCSRR0 (Save/Restore register)	Address of the instruction that caused the exception. Once the exception is finished (mcrfi instruction), program starts execution with the same instruction, that was the cause of the exception.

Machine check syndrome register (MCSR)

This register is the first register to check as it collects additional information about the cause of the exception. There are three groups of machine check causes

Table 3. Machine check causes

Machine check cause	Brief description
Error Report Machine check (IF,LD,ST,G)	These exceptions are directly associated with the current instruction execution stream. They are not masked with MSR_{ME} bit. It means the exception is always taken whenever the condition occurs. They enable to differentiate between Instruction fetch, Data store and load.
Non-maskable interrupt (NMI)	Not MSR_{ME} gated exception that occurs when NMI signaling is enabled and NMI pin is driven low.
Asynchronous Machine check (BUS_IREER, BUS_DRERR, BUS_WRERR)	Exceptions reported by the subsystem, usually as bus error termination, back to the Core. They have to be enabled by MSR_{ME} bit. They are cumulative. This machine check exception group triggers capture of the corresponding address to the MCAR register if $MCSR_{MAV}$ bit is cleared. If $MCSR_{MAV}$ was previously set, then the MCAR register is not affected.

Machine check address register (MCAR)

MCAR register contains target address reporting the fault condition. It is updated only for Asynchronous Machine check group when MCSR.MAV bit is cleared and it is valid only if MCSR.MAV status flag is set. Otherwise the MCAR register cannot be used in the fault analysis.

It is important to clear MCSR.MAV bit after reading MCAR register value to enable capture of the address in case of new asynchronous machine check fault.

Machine check MCSRR0 register

This register is updated by the HW in the beginning of the machine check interrupt. It stores the address of the instruction that caused the error condition.

It is used in the end of the machine check when *mcrfi* instruction is executed to fill the instruction pointer. The result is that code restarts the same instruction that was cause of the error, if additional modification of the MCSRR0 register is not explicitly done.

2 Machine check handler

Machine check handler usually splits into two parts:

- Low level handler
- User handler

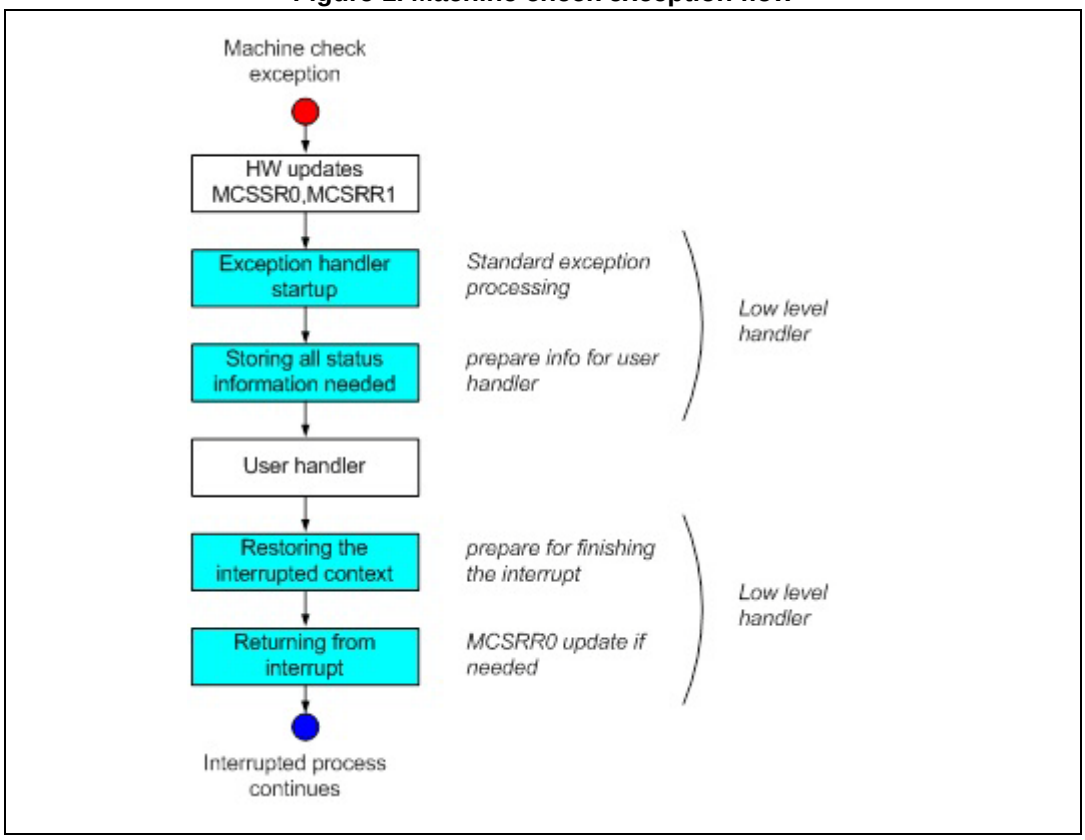
2.1 Low level handler

Low level handler is responsible for first and last part of the exception execution. It is usually written in assembly as it needs to execute proper instruction sequence before it can pass the code execution to higher level routine and accesses special purpose Core registers.

The middle of the interrupt service routine belongs to the user handler where analysis of the root cause of the exception and fault removal is done.

Once the user handler is finished code execution is given back to the low level driver to finish the interrupt and return back to the interrupted process.

Figure 1. Machine check exception flow



2.1.1 Start phase

First steps when machine check exception occurs are done by the hardware (Core) which stores content of the MSR register and address of the current instruction pointer if it is possible (precise exception).

Low level driver immediately starts processing after. It executes several steps like machine check status register saving, context of the interrupted process saving and other. This part should store some additional information too as they will be used by higher layer user handler to analyze the root cause of the exception later.

Detailed description of the machine check resources, their meaning and proper handling in case of interrupt is described in the Z4 Core User Manual documentation. Low level handler shall follow rules and recommendation described there.

2.1.2 Final phase

Here the handler should restore the saved context of the interrupted process and return with the mcrfi instruction.

Before *rfmci* instruction is executed, which fills instruction pointer with MCSRR0 content and MSR register with MCSRR1 content, MCSRR0 modification might be needed.

There are two cases which determine if the manipulation is needed or not, such information should be decided in the user handler and passed down to the low level driver

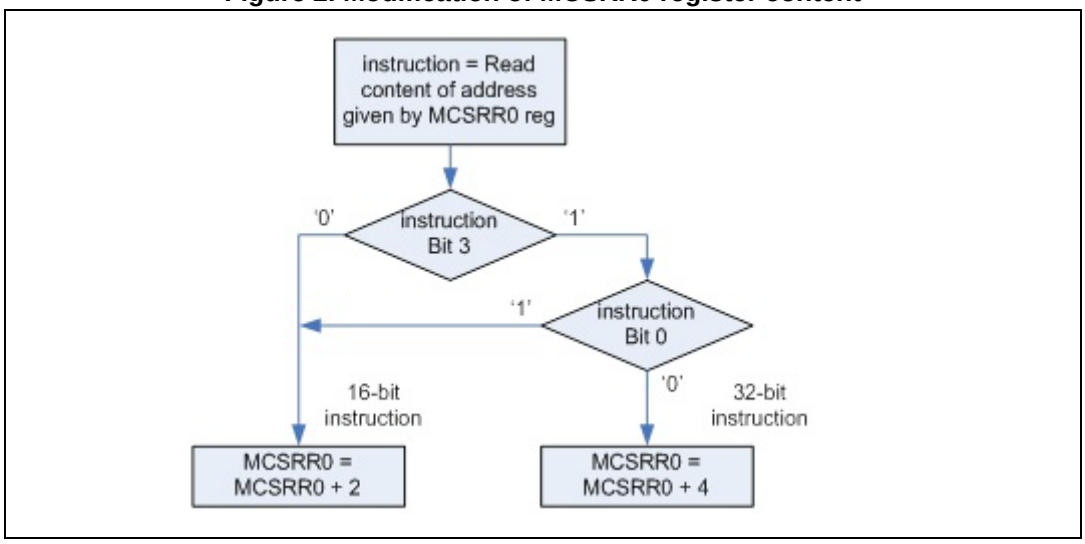
1. User handler was able to find the cause of the machine check exception and to fix it in a way, that program can re-execute the same instruction that caused the machine check exception
2. User was able to find the cause of the exception, but the problem remains and re-executing the same instruction would lead again the machine check exception => Modification of the MCSRR0 is needed.

2.1.3 Modification of the MCSRR0 register

In case that the cause of the exception cannot be removed, MCSRR0 register value has to be modified in a way that it takes the address of the following instruction. This prevents re-execution of the faulty instruction and retriggering the machine check exception.

Modification has to consider VLE instruction coding in case the interrupted process is implemented in VLE coding and increment the value accordingly of the length of the faulty instruction pointed by the current MCSRR0 register content, see *Figure 2*.

Figure 2. Modification of MCSRR0 register content



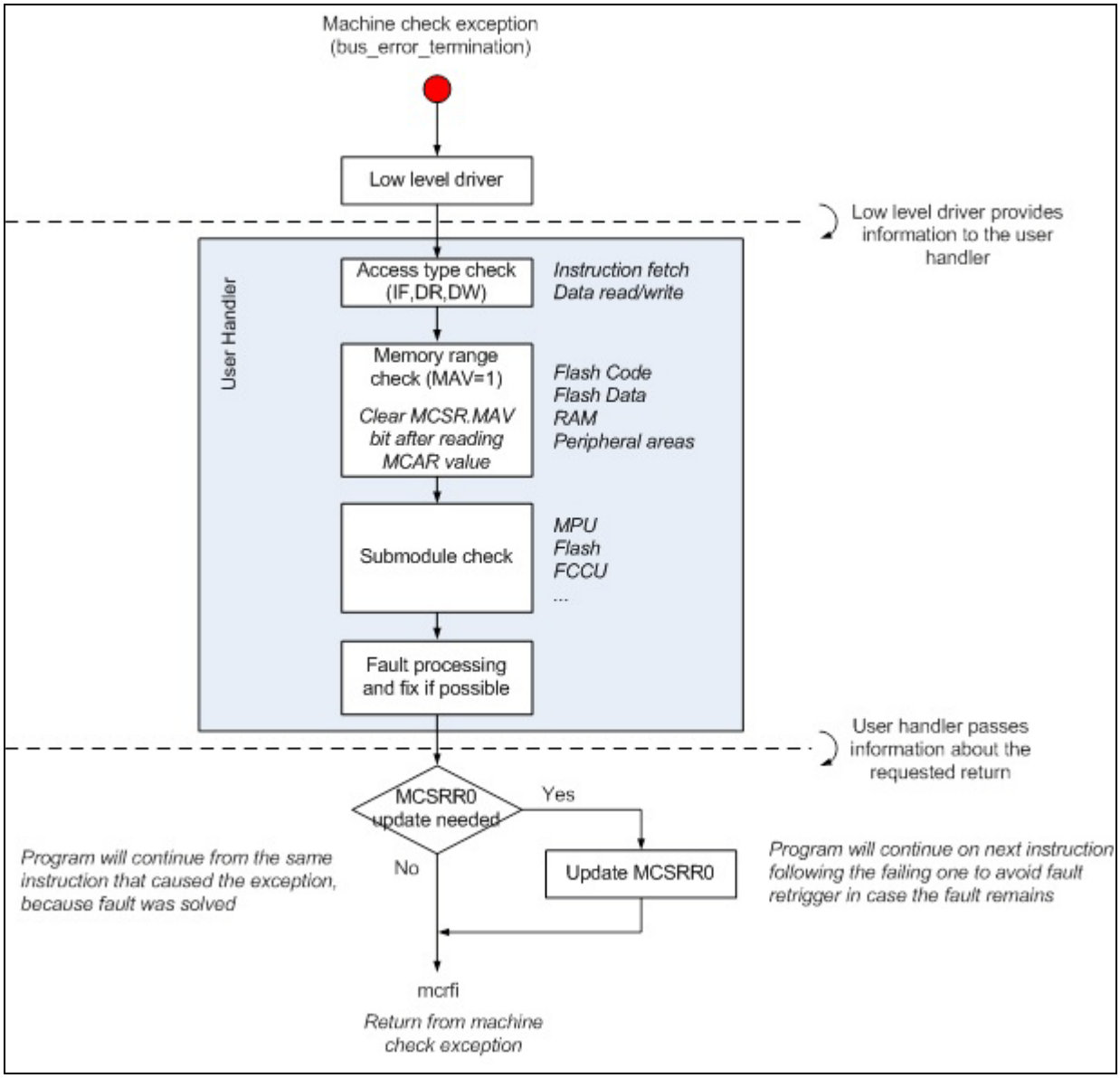
2.2 User handler

Here the root cause analysis is done. Such analyses requires supportive information from

- Low level driver (MCAR, MCSR etc.)
- Peripherals status registers for further elaboration

Based on the results of analysis and corrective actions done, user handler should pass the information about the return type back to the low level driver; indication if MCSRR0 content should be modified or not before *mcrfi* instruction.

Figure 3. Machine check exception user handler flow



3 SPC56EL60xx/SPC56xL70xx exception cases

This chapter lists most common exception cases that application software can experience while running code on SPC56EL60xx/SPC56xL70xx device.

Table 4. SPC56EL60xx/SPC56xL70xx exception causes

Exception cause	Error signaling	Exception	Description
Flash 2b ECC error	External bus error	Machine check	Two or multiple bit error in the Flash memory leads to the machine check exception when faulty area is read, instruction fetch or data read.

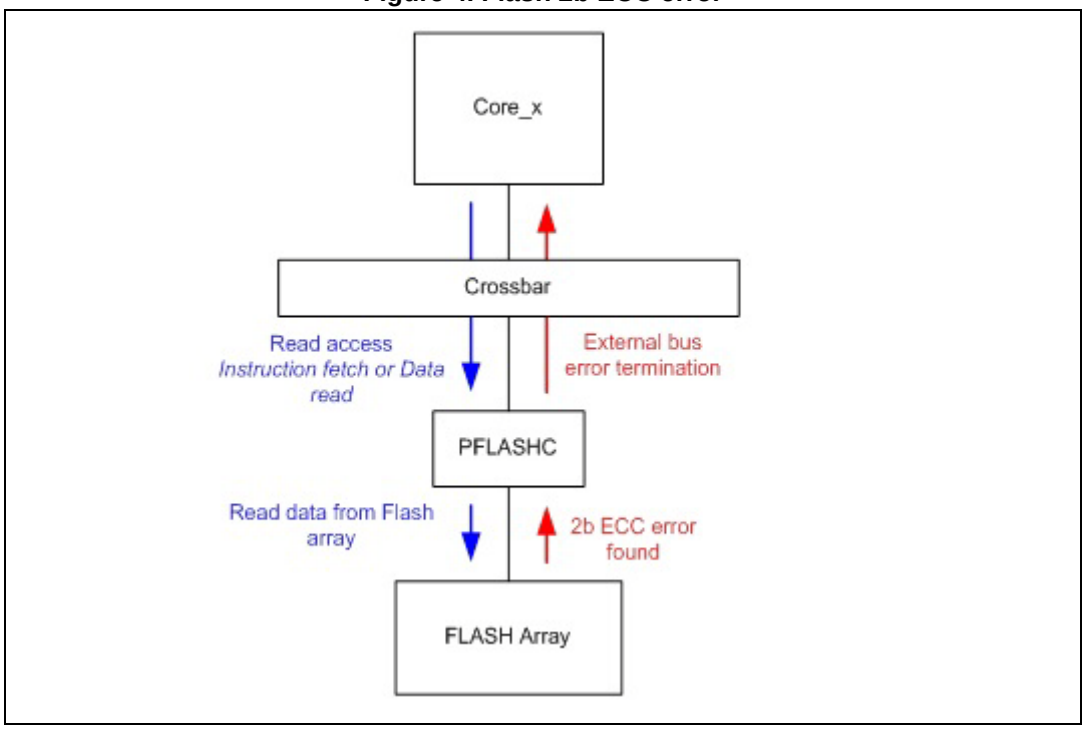
In general all protection access exceptions and 2b ECC exception leads to the same machine check exception because of external bus error termination. In such case further analysis relies on memory area check.

3.1 Flash 2b ECC error

3.1.1 Cause of the exception

Platform flash memory controller (PFLASHC) terminates bus transaction between CPU and PFLASHC controller in case the Flash memory array signals 2b ECC problem during read access. This leads to machine check exception because of *bus_error* termination.

Figure 4. Flash 2b ECC error



3.1.2 Machine check exception status

Table 5. Flash 2b ECC - machine check exception status

Register	Description
MCSRR0	Address of the instruction that caused the exception. In case of ECC error in the data flash area, register modification will be most probably needed.
MCSR	Type of operation is highlighted here, instruction fetch, data load or data write.
MCAR	Target address that was accessed, but finished with 2b ECC error. This address can be used for further analysis.

3.1.3 User exception handler

Handler has to analyze:

- Type of access, instruction fetch, data read, data write. Only instruction fetch or data read access are expected in case of 2bECC Flash error
- Memory range
- Memory access must be within area belonging to the Flash memory. User has to know which part belongs to the code flash and which part to the data flash memory.

3.1.4 Error solving

Flash 2b ECC error can be solved only with erase of the flash sector containing the cell with 2b ECC error. It is usually not the thing to be done in the exception handler itself, because it takes significant amount of time.

It is application specific the decision what to do in case of 2b ECC error, if to go to degraded mode or to continue, the case of EEPROM emulation, and to solve the issue later in the application.

If the decision is to continue, user handler has to request modification of the MCSRR0 register to continue the program flow with next instruction. Otherwise program would be stuck in the read of the fault flash address invoking machine checks.

Revision history

Table 6. Document revision history

Date	Revision	Changes
02-Oct-2013	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com